A Genetic Algorithm for the Mixed Flow Shop Problem

Master Thesis

Mario Pascual Poch

Prof. Matteo Brunelli

UNIVERSITY OF TRENTO

Department of Industrial Engineering

Master in Industrial Engineering

July 2019

Contents

CX Cycle Crossover Operator

D Dimension or number of pieces

ERO Edge Recombination Operator

M Machine stations

Mm Mutation Method

NCc Number of Crossover creatures

NMc Number of Mutation creatures

PS Population Size

SCX Sequential Constructive Operator

UX2 Union Crossover 2 operator

UX2 v2 Union Crossover 2 version 2 operator

WA Waiting Area stations

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis we present a new interesting version of the mixed flow shop sequencing problem, which at the same time is a version of the classic flow shop, a very common topic on operations research.

We propose a genetic algorithm to solve it that we will compare at the end with a simple initial genetic-based algorithm previously design. For that we first focus on the crossover operator as we consider it the most challenging part on a sequencing problem. We study and compare 5 different crossover operators and we choose the one that performs better. Finally we calibrate the population size, the weight of mutation and crossover operators on the algorithm and also the mutations operator itself.

The goal of the thesis is to better understand the specific mixed flow shop problem version presented and design a genetic algorithm that clearly improves the performance of the initial algorithm.

Chapter 2

The mixed flow shop problem

2.1 Definition

The mixed flow shop problem is a mix between the classic flow shop and the No wait flow shop.

On the one hand, the **classic flow shop** problem is a sequencing problem in which N pieces have to be processed on M machines or work stations. Every piece has to be processed only once on each machine and every machine can process only one piece at a time. For every piece i and machine j there is a specified processing time $t_{i,j}$ and the goal of the problem is to minimize the completion time of the process, also called makespan. That is the time between the beginning of the execution of the first piece on the first machine and the completion of the last job on the last machine. A solution of the problem is therefore a sequence of pieces. In fact every single sequence is a feasible solution. All pieces have to go through the M machines in the same order, which is the lexicographical order, that is $1, 2, 3, ..., j, ..., M$. So in order that the piece i can enter to the machine j two conditions must be satisfied:

1. The work on the machine $j-1$ for the piece i has to be already done. Or equivalently, a piece must have gone through all the machines before j to enter the machine j.
2. The work on the machine j for the piece $i-1$ has to be already done. Or equivalently, all the pieces before i must have gone through the machine j before i enters the machine j.

On the other hand, the **No wait flow shop** is a particularization of the classic

flow shop in which a restriction is added: all pieces have to be processed without idle times or interruptions between machines. That means that when a piece leaves a machine it has to immediately enter the next machine. Therefore a solution like the one represented in Figure 2.1 will not be feasible.

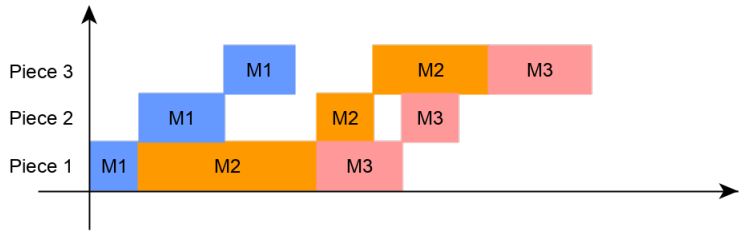

Figure 2.1: Unfeasible solution

Instead a feasible solution is shown in Figure 2.2:

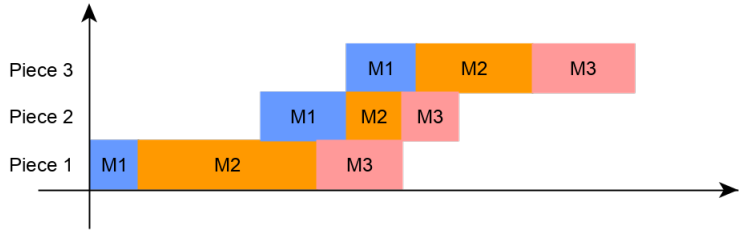

Figure 2.2: Feasible solution

So we can see that even though the second piece could start before according to the two conditions of the classic flow shop presented above, with the no wait condition it has to wait until it can be processed without idle times between machines. Note that all sequences are still feasible but now because of the no wait condition more of them will probably have worse completion times.

The **mixed flow** shop therefore is a sequencing problem in which some work stations have the no wait condition and others do not. Imagining an industrial process we will call the work stations with the no-wait condition as **Machine** stations and the ones without the no-wait condition as **Waiting Area** stations. Remember that the only difference between the machine stations and the waiting area stations is that the second ones can finish before the next working

station is available. In this case they leave the station empty for the next piece *i+1* while waiting the next station to be available for the piece *i*.

Another way to understand these waiting area stations is as if they were machine stations alongside unlimited-capacity storages. Then, when the work on the waiting area station is done, the piece can wait for the next machine to be available on the alongside storage. With the unlimited capacity condition we allow several pieces to stay at the same time waiting for the next machine. In section 2.3 we will show some applications that justify the interest of this approach.

2.1.1 Particular case

On this thesis we want to study a special case of the mixed flow shop problem. Instead of defining a specific processing time for each piece on each machine we define an interval of valid times. So in this particular case, for each piece i and machine j there will exist a minimum processing time called $tin_{i,j}$ and a maximum processing time called $tax_{i,j}$ and for each piece i and waiting area h there will be a minimum time called $sec_{i,h}$. Note that there is no maximum time for the waiting area stations as they were defined indeed without the no wait condition.

That means that the processing time of the piece i on the machine j can be between the minimum time $tin_{i,j}$ and the maximum time $tax_{i,j}$ and similarly the processing time of the piece i on the waiting area h has to be greater than the minimum time $sec_{i,h}$. If the processing time is within this range of values, the quality of the process according to the final purpose of the product is considered acceptable.

It can be that some processes for its characteristics need a specific processing time, then $tin_{i,j} = tax_{i,j} = k$ where k is any positive time value. Similarly, it can also be that some piece i does not need to be processed on the machine j, then $tin_{i,j} = tax_{i,j} = 0$.

The main reason of using these intervals of valid times is that, by introducing some flexibility, we make the problem more similar to what happens on the industry and therefore more realistic.

2.2 Previous Studies

Since the first flow shop problem was described and studied in 1953 by S. M. Johnson [9] hundreds of articles and papers have been published about different variations or particularizations of the same problem.

As seen before, the mixed flow shop problem is a particularization of the flow shop that combines the classic flow shop with the no-wait flow shop. In fact, every single modification to the original flow shop creates a totally different problem. For this, it is really interesting the work that Emmons and Vairaktarakis [6] did by organizing in one place a huge body of flow shop knowledge. They described for example some of the different and currently studied flow shop versions: The two-machine, the m-machine, the hybrid or the No-wait flow shop among others. So it can be considered a starting point to easily identify what is known or have been already studied in the literature. However we do not find any reference to our version in this work that was published in 2013.

Therefore to locate our problem in the current literature we have to turn to more recent literature, and for this, first we have to observe which characteristics or conditions our problem has that makes it different from the original flow shop. These characteristics are:

- No-wait stations combined with normal stations.
- Interval of valid times instead of a single time value.

The first characteristic seems to be found in the literature for the first time in 2018 [18] and it was defined as mixed flow shop, as we have been already referring to the problem. Much earlier the No-wait in process flow shop was described for the first time by S.S. Reddi and C.V. Ramamoorthy in 1972 [16].

The second characteristic though, to the best of our knowledge, has never been considered. Some studies have considered similar cases such as variable processing times taking into account learning and deteriorating effects [19], or stochastic processing times [15] [12] but never intervals of valid times.

This second condition therefore makes this problem an unstudied version of the flow shop problem, but still pretty similar to the mixed flow shop version.

2.3 Applications

The main reason to study this new flow shop problem version is because of its direct applications in the industry. There are a lot of processes in which both no-wait stations (machines) and non no-wait stations (waiting areas) coexist. For example R. Ruiz, S. Sui, Y. Wang and X. Li described with great level of precision a couple of examples [18]:

"*In the canned food processing industry, no-wait is not needed for many operations such as purchasing, classification, pruning, cleaning, and removing the peel and shell. On the contrary, no-wait is required for the following operations:*

adding sugar liquid, gas exhausting, sealing, sterilizing, and refrigerating once the food is precooked and while it is still hot. As a result the no-wait constraint is not needed again for subsequent operations such as the labeling, handling, palletizing, etc. because the food has been preserved safely in cans. Another typical example is producing mannitol from starch. Once size-mixing starts, no wait operations follow immediately (the first jet liquifying, liquifying in a reaction jar, the second jet liquifying, refrigerating, adjusting PH value and saccharifying). Any wait in between two operations would result in the starch becoming thick and then solid after it is heated and size mixed. However, waiting is permitted in later operations such as concentrating, separating and crystallizing."

Also this problem is very interesting in classic no-wait flow shop problems like plastic, steel, chemical, pharmaceutical, food processing or concrete production [11], [8], [2] in which we can easily consider some stations such as cooling or drying machines or even rest stations or stocks that can be modeled as waiting area stations. That was stations without the no-wait condition or with a minimum time and not a maximum one.

Some specific examples of this kind of stations can be a drying station after a painting machine or an unforced cooling after a welding station or after an exothermic chemical reaction, among others.

On the other hand, the intervals of times were defined to better represent some processes in which the desired condition is achieved from one specific moment but also until another specific moment. Some examples of these processes are: warming or cooling processes, chemical reactions, cleaning processes, products mixture processes or liquid immersion processes.

Chapter 3

Completion time of a sequence

To design an algorithm that solves a flow shop problem it is first necessary to know how to calculate the completion time of a sequence. In the regular flow shop problem this consist in adding processing times. On our particular case, however, we do not have specific times but intervals of time, and some stations have a maximum time (machines) and others do not (waiting areas). For all this it is worth explaining how to calculate the completion time of a sequence as it is not obvious at first sight.

This algorithm starts by dividing the problem in k sub-problems, where k equals the number of waiting areas plus one. Each of these k sub-problems consist in the portion of the main problem between two waiting areas (including the one at the end). So for example if the process consist of:

$$M_1 \to M_2 \to M_3 \to WA_1 \to M_4 \to M_5 \to WA_2 \to M_6 \to M_7 \to M_8$$

Where $\boldsymbol{M}$ are the work stations of the type machines and the $\boldsymbol{WA}$ the work stations of the type waiting areas. Then we would study for all the pieces, first the sequencing from M_1 to WA_1, then from M_4 to WA_2 and finally from M_6 to M_8.

On each of these stages the procedure is the following:

1. Schedule the first piece with its minimum times in all the machines of the stage. See Figure 3.1.
2. Schedule the piece i on the machine j starting on the maximum time between the time in which the piece $i-1$ finishes on the machine j and the time in which the piece i finishes on the previous machine $j-1$ (in case there is a previous machine). See Figure 3.2.

3. If the piece i on the machine j finishes before the piece $i-1$ on the machine $j+1$ apply 3.1, 3.2 or 3.3 according to the situation in order to satisfy the following condition: That the piece i on the machine j finishes at the same time as the piece $i-1$ on the machine $j+1$.

   3.1. If the station before j is a machine or there is no station before, extend the duration of the piece i on the machine j until the condition is satisfied as long as its maximum time allows it. See Figure 3.3.

   3.2. If the station before j is a machine but its maximum time does not allow to satisfy the condition: Extend the duration of the piece i on the machine j until its max time and delay the beginning of machine j (for j=1) or the beginning of the set of machines *j, j-1,. . . ,1* (for i>1) for the piece i until the condition is satisfied (keeping the processing times on those delayed stations). See Figure 3.4.

   3.3. If the station before j is a waiting area, extend the duration of the waiting area until the condition is satisfied. See Figure 3.5.

4. Next station, j=j+1. Return to (2) until the last machine of the process is reached.

5. Schedule the waiting area of the piece i with its minimum time.

6. Next piece, i=i+1 and go back to machine m=1. Return to (2) until the last piece is reached.

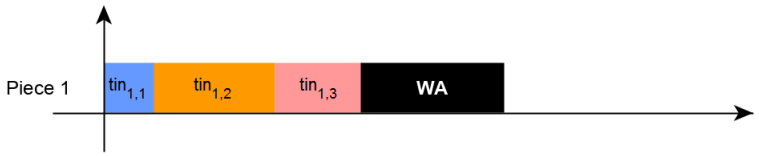

Figure 3.1: Visual representation of step 1 of the completion time algorithm. The first piece is scheduled in the first group of machines and waiting area with its minimum times.

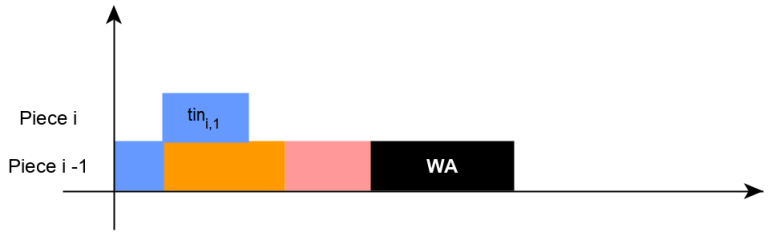

Figure 3.2: Visual representation of step 2 of the completion time algorithm. The second piece is scheduled in the first machine as soon as possible.

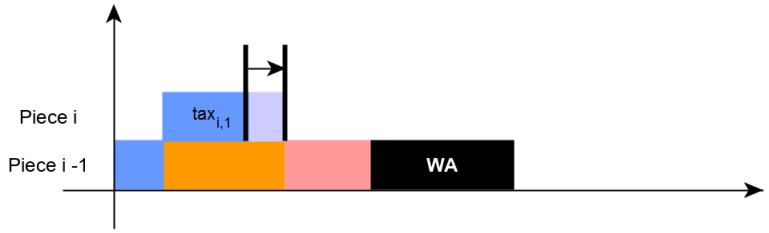

Figure 3.3: Illustration of a situation in which condition 3.1 applies

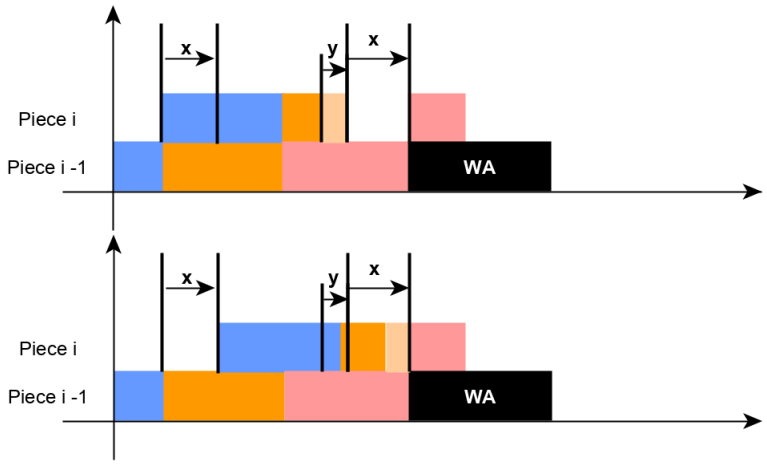

Figure 3.4: Illustration of a situation in which condition 3.2 applies. First we extend the duration of the piece i to its max (represented by y) and then we delay all the previous pieces as long as needed (represented by x).

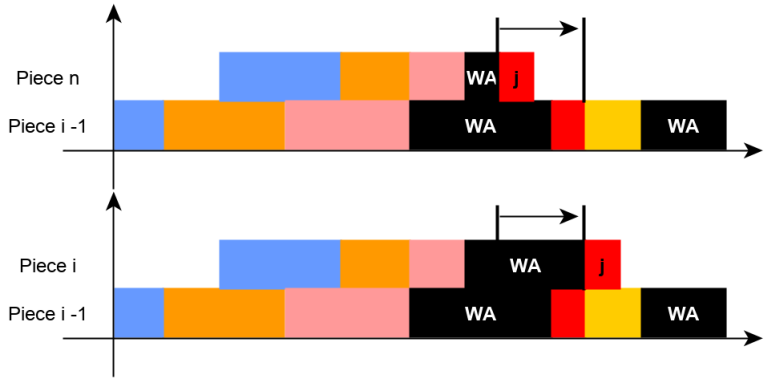

Figure 3.5: Illustration of a situation in which condition 3.3 applies

Chapter 4

First algorithm version

4.1 How it works

The first version of the algorithm was conceived on a project carried out in one of the last subjects of the Master degree on industrial engineering. On that project I developed and coded a first genetic based algorithm to find a good solution for the mixed flow shop problem.

First of all the data of the problem is presented on a txt file according to the following format:

N
M
H
$tin_{1,1} * tin_{1,2} * \ldots * tin_{1,M}$
$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$
$tin_{N,1} * tin_{N,2} * \ldots * tin_{N,M}$
$tax_{1,1} * tax_{1,2} * \ldots * tax_{1,M}$
$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$
$tax_{N,1} * tax_{N,2} * \ldots * tax_{N,M}$
$b_1 * b_2 * \ldots * b_H$
$sec_{1,1} * sec_{1,2} * \ldots * sec_{1,H}$
$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$
$sec_{N,1} * sec_{N,2} * \ldots * sec_{N,H}$

where N is the number of pieces, M the number of stations of the type machines, H the number of stations of the type waiting areas, *tin* and *tax* the minimum and maximum times on the machines, *sec* the minimum time on the waiting areas and b the position in which there is a waiting area station. For

example if b is 4 it means that there is a waiting area between the machines 4 and 5.

So the steps of the algorithm are the following:

1. Reading and decoding of the given data on the txt file.
2. Determination and calculation of 15 initial sequences.
   2.1. Determination of 9 sequences according to some logic criteria:
      - Sequence 1: In ascending order, the pieces with lower sum of minimum times ($tin_{i,j}$) on the M machines.
      - Sequence 2: In ascending order, the pieces with lower sum of maximum times ($tax_{i,j}$) on the M machines.
      - Sequence 3: In ascending order, the pieces with lower sum of the average of max and min times ($(tin_{i,j} + tax_{i,j})/2$) on the M machines.
      - Sequence 4: In ascending order, the pieces with lower sum of minimum times ($tin_{i,j}$) on the first half of the M machines.
      - Sequence 5: In ascending order, the pieces with lower sum of maximum times ($tax_{i,j}$) on the first half of the M machines.
      - Sequence 6: In ascending order, the pieces with lower sum of the average of max and min times ($(tin_{i,j} + tax_{i,j})/2$) on the first half of the M machines.
      - Sequence 7: In ascending order, the sum of the minimum initial time ($tin_{i,1}$) with the minimum times of the machines immediately after the waiting area stations.
      - Sequence 8: In ascending order, the sum of the maximum initial time ($tax_{i,1}$) with the maximum times of the machines immediately after the waiting area stations.
      - Sequence 9: In ascending order, the sum of the average initial time ($(tin_{i,j} + tax_{i,j})/2$) with the average times of the machines immediately after the waiting area stations.

   2.2. Calculation of the completion time of the 9 logic sequences.

   2.3. Calculation of the completion time during 25 seconds of random sequences. The program keeps the best 15 taking into account both the ones calculated in (2.2) and these random sequences.
3. Main loop. Running time set up at 290 seconds from the initialization of the program.
   3.1. Loop for every one of the 15 sequences.

3.2. Random selection between algorithm 1 or algorithm 2.

3.3. If algorithm 1:

3.3.1. For each piece i of each of the sequence we define a parameter called $lgap_i$ which is the sum of the differences between the scheduled times on the waiting area stations and the minimum time in those stations ($sec_{i,h}$). That is $\sum_{h=1}^{H}(Tout_{i,h} - Tin_{i,h}) - sec_{i,h}$, where $Tout_{i,h}$ and $Tin_{i,h}$ are the times in which the piece i leaves and enters the working area h.

3.3.2. We look for the 2 pieces associated with the 2 highest $lgap_i$ values.

3.3.3. From the original sequence we define 16 new sequences:

- Sequences 1-4: For each of the two pieces with highest $lgap_i$ value, we apply an alteration to the original sequence consisting of permuting that piece for the one immediately before and the one immediately after. Two new sequences for each $lgap_i$ value so 4 new sequences in all.
- Sequences 5-8: From the original sequence one alteration is made between two consecutive pieces. The two pieces of the alteration are decided randomly. This is done 4 times (not sequentially, in parallel).
- Sequences 9-12: From the original sequence one alteration is made between two pieces, in this case, not necessarily consecutive. The two pieces of the alteration are decided again randomly. This is done 4 times.
- Sequences 13-16: From the original sequence three elements are randomly chosen and permuted. This is done 4 times.

3.3.4. We calculate the completion times of the previous 16 sequences. If one completion time is better than the worst of the 15 best sequences find at the moment, we keep the sequence on a provisional list.

3.4. If algorithm 2:

3.4.1. For each piece i of each of the 15 sequences we define a parameter called $lgap2_i$ which is the sum of the differences between the real times on the machines and the minimum time in those stations. That is $\sum_{j=1}^{M}(Tout_{i,j} - Tin_{i,j}) - tin_{i,j}$, where $Tout_{i,j}$ and $Tin_{i,j}$ are the times in which the piece i leaves and enters the machine j.

3.4.2. We look for the 2 pieces associated with the 2 highest $lgap2_i$ values.

3.4.3. From the original sequence we define 16 new sequences:

- Sequence 1-4: For each of the two pieces with highest $lgap2_i$ value, we apply an alteration to the original sequence consisting of permuting that piece for the one immediately before and the one immediately after. Two new sequences for each $lgap2_i$ value so 4 new sequences in all.
- Sequences 5-8: From the original sequence one alteration is made between two consecutive pieces. The two pieces of the alteration are decided randomly. This is done 4 times (not sequentially, in parallel).
- Sequences 9-12: From the original sequence one alteration is made between two pieces, on this case, not necessarily consecutive. The two pieces of the alteration are decided again randomly. This is done 4 times.

3.4.4. We calculate the completion times of the previous 12 sequences. If one completion time is better than the worst of the 15 best sequences find at the moment, we keep the sequence on a provisional list.

3.5. Finally we update the *15 best sequences* list with the best ones among the current 15 best sequences and the ones on the provisional list.

Figure 4.1 represents the process visually:

4.2 Calibration

In this algorithm, where the running time was fixed to 290 seconds, it is obvious that the more sequences we generate per iteration the less iterations there will be. Therefore the three main parameters that had to be adjusted were indeed the number of saved sequences or population size and the number of generated neighbours from algorithm 1 and from algorithm 2.

After analysing a few versions by testing 100 cases between 20 and 60 pieces, we set up those parameters to 15 saved sequences, 16 generated neighbours from algorithm 1 and 12 generated neighbours from algorithm 2 with the peculiarity that on every iteration only one algorithm would be applied (as explained in 4.1) to reduce the calculation time of every iteration without having to stop using any of the two algorithm variations.

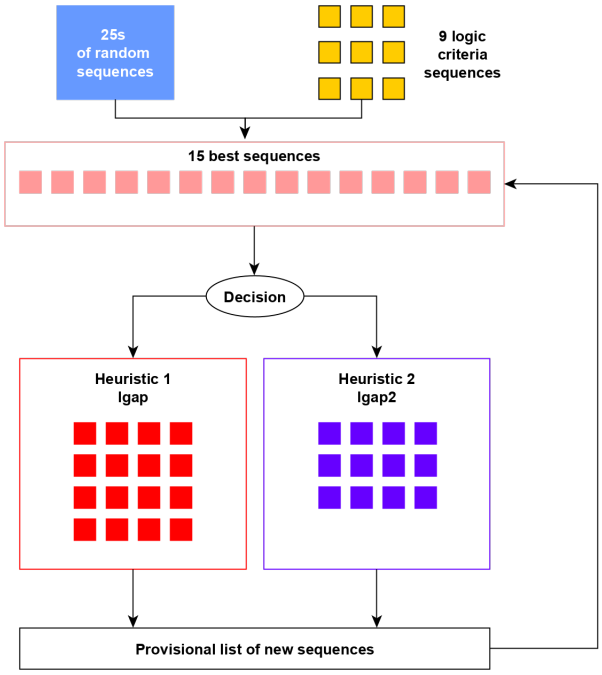

Figure 4.1: Visual representation of the algorithm

Chapter 5

New genetic algorithm

5.1 Introduction to genetic algorithms

Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics [4]. An initial population of creatures (also called sequences in this problem) evolves through crossover, mutation and selection to a better population of species according to some survival criteria. It is an iterative process in which each iteration is called generation. In each generation a new set of creatures is created from the current creatures and some of them also from mutation. Then the fitness of the new creatures or offsprings is evaluated with an objective function and finally some of them are selected for the next generation according to its fitness value.

So the different parts of a genetic algorithm answers the following questions:

- **Initial population** and **Population size**: How do we generate the initial population and how big this has to be? The bigger it is the more space of solutions we will explore on each iteration, but at the same time, the longer it will take each generation to compute.
- **Crossover**: How do we generate offsprings from the current population? To guarantee an evolutionary process the offsprings should carry information of their parents.
- **Mutation**: How often random mutations take place and what they consist on? Mutation prevents the algorithm to be trapped in a local minimum. It is also viewed as a background operator to maintain genetic diversity in the population [13].
- **Selection**: Which is the fitness criteria to determine which creatures are better and how we determine which ones go trough the next generation

and which ones die?

- **Termination condition**: When will the algorithm stop running?

5.2 Crossovers

The most challenging part of our genetic algorithm is, without a doubt, the crossover. The crossover operator must transfer information from some creatures to their offsprings.

Most genetic algorithms work with binary strings creatures so that every position or group of positions on the string is related to some characteristic of the solution. On this problem, instead, the creatures are a sequence of numbers in which all numbers are present once and just once. That means that most of the simplest crossover methods usually used on genetic algorithms are not useful on this particular problem.

For example take the simplest but most used crossover operator, the single point crossover. It consist on randomly define a point on the string of both parents and then generate two offsprings by mixing the right information of parent 1 with the left information of parent 2 and vice versa. We can easily see that if we apply this operator to our problem we most likely get a non valid offspring. Considering a n=5 problem (so with 5 pieces) and the following two parents P1: 12345, P2: 52314, with the crossover point between the second and the third node or element. A **node** or **element** is each number of the sequence that remember represents a different piece.

P1: 12|345
P2: 52|314

Their 2 offsprings would be:

O1: 12314
O2: 52345

These two offsprings are not possible as the sequence has to contain all the pieces once and just once.

At this point we present 4 different crossover operators from the literature and one new version of one of them. Then we will test them to see how do they work on this problem and we will see if they perform equally or some are better than the others.

5.2.1 ERO

The Edge Recombination Operator or **ERO** (also found as ERX in the literature) was proposed in 1989 for the travelling salesman problem [10]. It is an edge based crossover operator which means that rather than looking at the nodes itself it looks at the edges or relations between the nodes of the parents. From two parents it creates a new offspring with characteristics (edges) of both parents.

The way it works is the following. We will consider the two parents P1: 12345 and P2: 51432 to illustrate the algorithm :

1. Creation for each parent of an adjacency matrix. That is, a list for each node with its neighbours nodes. The first an last node of a sequence are also considered neighbours.

From P1	From P2
1:[2,5]	1:[5,4]
2:[1,3]	2:[3,5]
3:[2,4]	3:[4,2]
4:[3,5]	4:[1,3]
5:[4,1]	5:[1,2]

2. Union for each node of both neighbours lists (without repeated elements).

   1:[2,4,5]
   2:[1,3,5]
   3:[2,4]
   4:[1,3,5]
   5:[1,2,4]

3. Randomly select the first node for the offspring from one of the parents first nodes.

   Offspring: 5,_,_,_,_

4. Remove this node from all neighbours lists.

   1:[2,4]
   2:[1,3]
   3:[2,4]
   4:[1,3]

5. While the length of the offspring is not equal to the length of the parents, select the node with less neighbours (so with less elements into its neighbour list) and remove it from all the neighbours list. In case of draw, chose randomly.

A possible offspring could be then 54213.

We can observe that besides the first nodes this crossover operator keeps also some of the parent nodes relations, which seems useful for this problem.

On the contrary, it considers the relation between the first and the last pieces which for the travelling salesman problem is fine but for a flow shop problem does not make sense. Also the output can be exactly the same as one of the inputs, especially for short sequences, which would not bring any progress to the evolutionary process. Nevertheless, the longer the sequence the less probable this is to happen.

5.2.2 CX

The Cycle Crossover Operator or **CX** was first proposed in 1985 by I.M. Oliver [7]. The main characteristic of this algorithm is that each elements of the offsprings share its position with at least one of the parents.

As in the previous case, we will explain how this algorithm works with the same example, P1: 12345 and P2: 51432. For this operator it is very important the position in which the different elements are on the sequence so we will use the notation $P2(x) = y$ to indicate that the element of P2 in the position x is y:

1. Select the first node from P1.

   Offspring: 1,_,_,_,_

2. Look for $P2(1)$ in P1. Suppose that $P2(1) = a$ and $P1(b) = a$ and write the element a on the offspring on the position b.

   The first element in P2 is 5 that happens to be in fith position in P1. So we write 5 in fith position.

   Offspring 1: 1,_,_,_,5

3. Then again look for $P2(b)$ in P1. Suppose that $P2(b) = c$ and $P1(d) = c$ and write the element c on the offspring on the position d. Repeat this loop until the new found element is already in the offspring. At this point we will have found a set of elements that share the same set of positions on both parents.

   The element in fifth position in P2 is 2, which in P1 is found in the second position, so we put the 2 in this position.

   Offspring 1: 1,2,_,_,5

Now the element in second position in P2 is 1, which is already on the set, so we stop. Note that indeed the elements 1, 2 and 5 are found either first, second or fifth position in both parents.

4. Complete the empty spots in the Offspring with the elements of P2 in those positions.

   Offspring 1: 1,2,4,3,5

5. Repeat the process changing P1 for P2, so select this time the first node from P2.

   In this case the first set from P2 would be:

   Offspring 2: 5,1,_,_,2

   And completed with the elements of P1:

   Offspring 2: 5,1,3,4,2

This algorithm ensures that all the elements of the offsprings share their position with at least one of their parents. Notice that it is possible to end up with the offsprings being the same as the parents. But again the longer the sequence the less probable this is.

5.2.3 SCX

The Sequential Constructive Crossover operator or **SCX** was developed by Zakir H.Ahmed on 2010 [1] for the travelling salesman problem in which it performed better than the ERO operator. According to Zakir, the sequential constructive crossover operator constructs an offspring from a pair of parents using better edges on the basis of their values that may be present in the parents' structure maintaining the sequence of nodes in the parent elements.

Again we will explain how this algorithm works following the same example, P1: 12345 and P2: 51432. But first we need to define some concepts.

A **legitimate node** of a certain node is the next node in a sequence that is not yet included on the offspring. So for example if the sequence is 12345 and the current in construction offspiring is 2,1,_,_,_ , the legitimate node of 1 is 3, because it is the first node after itself not yet included in the offspring .

The **cost matrix** c is a nxn matrix clearly thought for the travelling salesman problem, in which each element $c_{i,j}$ represents the distance or the cost between the cities i and j. As we don't have this concept of distances or cost between pieces (the equivalent of cities in the travelling salesman problem) we

had to adapt it and we came up with the following version for the flow shop problem: In this case the element $c_{i,j}$ represents the completion time of the piece i followed by the piece j. This gives us an equivalent information about how "difficult" is to go from piece i to piece j.

Now we can look at the algorithm:

1. Select a random node to start.

   Offspring: $2_{,\_,\_,\_,\_}$

2. Search on both parents the first legitimate node from the last included node on the offspring. If there is no legitimate node, take the first node of the list (1, 2, 3, ..., n) not yet included in the offspring.

   Legitimate node from 2 in P1: 3
   Legitimate node from 2 in P2: 1 (2 is the last elements in P2 so we have to take the legitimate node from the ordered set of unchosen nodes, which is 1)

3. If there is just one legitimate node, take it as the next node in the offspring. If there are two nodes i and j chose i so that $c_{k,i} < c_{k,j}$ where k is the last element of the offspring at the moment.

   If $c_{2,3} = 50$ and $c_{2,1} = 20$ the next node would be 1 as $c_{2,1} < c_{2,3}$.

4. Repeat steps 2 and 3 until the offspring is completed.

In the mentioned article [1] there is a full example that illustrates in more detail how this crossover works.

The SCX crossover operator performs excellently in the travelling salesman problem mainly because of the cost matrix. It has to be seen though if it works as well in the flowshop problem. What we can ensure by the moment is that it will be the most time consuming crossover operator, as it has to calculate the cost matrix, which means nxn 2-pieces subproblems.

5.2.4 UX2

The Union Crossover 2 operator or **UX2** comes from the UX, described by B. R. Fox and M. B. McMahon [5]. They reported that UX operator performed well when compared with other crossover operators. However, the computation time required was high compared to the others, that is why later P. W. Poo and J. N. Carter [14] proposed a modification called UX2 that, in their words, does exactly the same operations as UX but in significantly less computing time.

Again, following the example with the parents P1: 12345 and P2: 51432 this is how the UX2 algorithm works:

1. Chose randomly a substring of elements from P2 and write them to Substring 1 or S1.

   S1=[432]

2. Write the remaining elements of P2 to Substring 2 or S2 in the order in which they appear in P1.

   S2=[15]

3. Randomly chose S1 or S2, write the first element of that substring in the offspring and delete it from the substring. Repeat this until one of the substrings is empty.

   Choose S2
   Offspring=1,_,_,_,_ S1=[432] S2=[5]
   Choose S1
   Offspring=1,4,_,_,_ S1=[32] S2=[5]
   Choose S2
   Offspring=1,4,5,_,_ S1=[32] S2=[]

4. Complete the offspring with the remaining elements on the non empty substring.

   Offspring=1,4,5,3,2 S1=[] S2=[]

The length of the first substring of P2 is not specified so it is suppose to be random. In this case however is defined as half the length of the parents strings.

5.2.5 UX2 v2

The UX2 crossover operator seems interesting as it creates the new offsspring from two parents substrings. However it seems much more interesting if instead of combining them randomly we combined them one after the other so that we keep a long unaltered substring from one of the parents.

That is what we propose in the new crossover operator, Union Crossover 2 version 2 or **UX2 v2**: Define and create in the same way both substrings S1 and S2 but then generate the offspring from combining directly S1 and S2. In fact we propose to generate two offsprings, S1 followed by S2 and S2 followed by S1.

Again, following the example from the parents P1: 12345 and P2: 51432 this is how the UX2 v2 algorithm works:

1. Chose randomly a substring of elements from P2 and write them to Substring 1 or S1.

   S1=[432]

2. Write the remaining elements of P2 to Substring 2 or S2 in the order in which they appear in P1.

   S2=[15]

3. Generate the two offsprings by concateneting S1 with S2 and vice versa.

   Offspring 1 =4,3,2,1,5
   Offspring 2 =1,5,4,3,2

We see that the offsprings are more similar with their parents as they keep a full substring from one of them, which we think can work well with a flowshop problem.

5.3 Crossover comparison study

To study how these crossover operators work on this particular mixed flow shop problem we propose a series of tests. First we define the parameters that we take into account:

- Number of machines. This is the less important parameter as it makes little difference from a computational point of view to have 10 or 40 machines
- D=Dimension of the problem or number of pieces
- C=Crossover operator
  0. ERO
  1. CX
  2. SCX
  3. UX2
  4. UX2 v2
- N=Number of repetitions or times that each combination is calculated to reduce the impact of randomness.
- WA=Number of Waiting Area stations

- Running time
- Population size
- Termination condition

5.3.1 Step 1-Is it possible to calculate all the possible combinations?

The first question that we should answer before starting to solve any flow shop problem is: Can we find the optimal solution? That is, the best possible one.

This can be done in two ways, either by using some algorithm that, for its procedure and characteristics, assure us the optimum, or by simply enumerating all the feasible solutions. Because it is a completely new problem, there is no such an algorithm. So the only possibility to find the optimum and assure that indeed it is the optimum is to look at all the different combinations. This looks with a naked eye impracticable as it is a factorial problem, which means that in a problem with n pieces there are n! possible solutions.

In the following Table 5.1 we can find the computing time to calculate all the combinations for the smaller problems (with less pieces). For a D=10 problem it takes more than 1 hour. We also prove in Figure 5.1 that there is an exponential growth so for example for D=15 the forecast is 10000000 seconds or 115,75 days. Therefore we prove that indeed it is impracticable to calculate all the possible solutions for bigger problems as it takes too much time. This and all the following calculations have been made with a personal computer with intel core i5-2410M and 4 GB DDR3 Memory.

D	**time [s]**
1	0,001
2	0,001
3	0,004
4	0,015
5	0,078
6	0,561
7	4,409
8	37,360
9	373,827
10	4329,574

Table 5.1: Computing time in relation to the dimension or number of pieces

Having seen that it is not practicable to go trough all the possible combinations on a problem with 10 pieces or more we can check if at least the genetic algorithm converges to the optimum reasonably fast. At this point we have 5 versions of

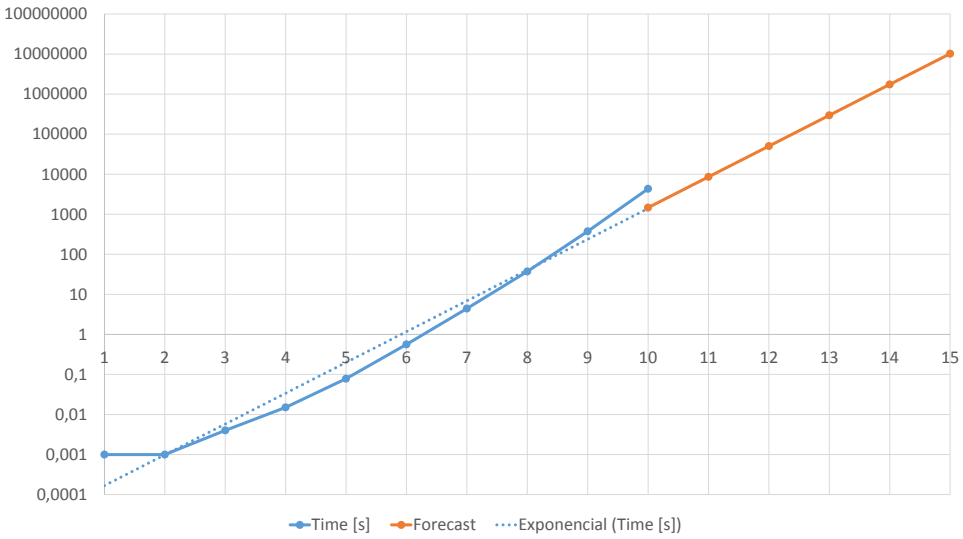

Figure 5.1: Graphic representation in logarithmic scale of the computing times to analyze all the combinations in seconds from D=1 to D=10 and forecast to D=15

the algorithm, each one with a different crossover operator. We will first test problems with D=7,8,9 and 10 pieces and for this and all the following tests, N (the number of repetitions for each combination of parameters) is set to 5 as there is a lot of randomness involved and we want to get robust results. So the shown results will always be the average of those 5 cases. See the test results in Table 5.2.

D	**C0**	**C1**	**C2**	**C3**	**C4**
7	0,948	1,148	3,152	0,524	0,522
8	1,770	1,251	8,734	1,629	1,573
9	4,586	2,991	18,885	2,005	2,219
10	6,264	5,097	67,477	4,688	3,285

Table 5.2: Computational times in seconds to get to the optimum

All crossovers except C2 seem to get to the optimum in a reasonably fast time but still we can not assure that the results found in bigger dimensions problems will be the optimums. What we can only say at this point with these results is that crossovers C0, C1, C3 and C4 seem to be faster than C2.

5.3.2 Step 2-Small dimensions. Are some of the crossovers better than others?

The problems studied before are not really interesting as the optimum can be simply obtained from looking at all the different combinations. That is why we move now into what we consider small dimensions, D=15,18,20,22 and 25. We will also try to see if there is any difference on the *WA* parameter, so we will consider *WA*=1 and 4 and of course the 5 different crossovers C0, C1, C2, C3 and C4. *N* as always will be 5 and the running time will be 120 seconds which is considered to be a reasonable time. There will be no other termination condition.

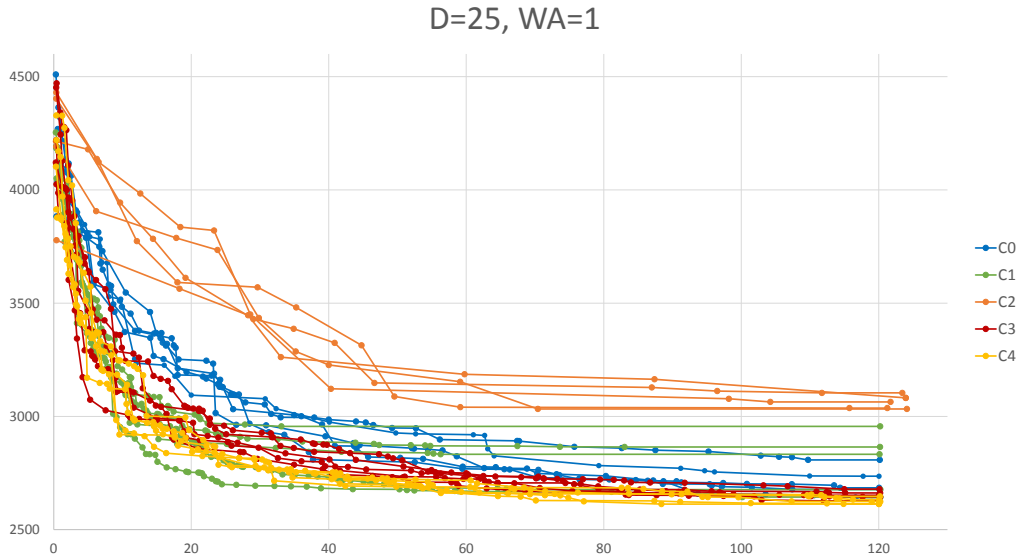

Figure 5.2: Evolution of the different crossover operators with a D=25 and WA=1 problem

In Figures 5.2 and 5.3 we can observe the evolution of the biggest test of the set, which is *D*=25 pieces. It represents the objective function that we are trying to minimize, the completion time in y, with computational time in x, that is set to 120s. Each point represents the finding of a new best makespan value found so far. As *N* was set to 5, there are 5 solutions for each crossover operator.

In Tables 5.3 and 5.4 we can observe for every dimension *D* and every crossover operator *C*, the best objective function result among the five solutions of each case (Best result) and the average of those (Average Result). This is interesting because we don't want only an algorithm that performs really well sometimes but one that is robust. In other words, that can always get an acceptably good solution. Then to compare how far or close are the different results for the

D=15					
C	**C0**	**C1**	**C2**	**C3**	**C4**
Best result	1986	1986	2054	2000	2000
Average result	1997	2055	2064	2000	2001
Deviation respect best result	0,00%	0,00%	3,42%	0,70%	0,70%
Deviation respect best average result	0,00%	2,87%	3,34%	0,14%	0,21%
D=18					
C	**C0**	**C1**	**C2**	**C3**	**C4**
Best result	2177	2165	2280	2167	2165
Average result	2190	2183	2310	2179	2170
Deviation respect best result	0,55%	0,00%	5,31%	0,09%	0,00%
Deviation respect best average result	0,90%	0,59%	6,45%	0,42%	0,00%
D=20					
C	**C0**	**C1**	**C2**	**C3**	**C4**
Best result	2260	2275	2443	2281	2257
Average result	2272	2326	2510	2286	2269
Deviation respect best result	0,13%	0,80%	8,24%	1,06%	0,00%
Deviation respect best average result	0,14%	2,51%	10,61%	0,75%	0,00%
D=22					
C	**C0**	**C1**	**C2**	**C3**	**C4**
Best result	2456	2468	2728	2447	2437
Average result	2478	2505	2764	2454	2446
Deviation respect best result	0,78%	1,27%	11,94%	0,41%	0,00%
Deviation respect best average result	1,32%	2,40%	13,00%	0,34%	0,00%
D=25					
C	**C0**	**C1**	**C2**	**C3**	**C4**
Best result	2642	2644	3033	2627	2613
Average result	2710	2795	3064	2653	2626
Deviation respect best result	1,11%	1,19%	16,07%	0,54%	0,00%
Deviation respect best average result	3,18%	6,43%	16,67%	1,02%	0,00%
Total					
Average deviation respect best result	0,52%	0,65%	9,00%	0,56%	0,14%
Average deviation respect best average result	1,11%	2,96%	10,02%	0,53%	0,04%

Table 5.3: Small dimension 1WA results

same dimension problem we define the deviation respect best result which is the relative distance between the best result obtained with each crossover and the best result among all 5 crossovers, and the deviation respect best average result which is the same but with the average results instead of with the best results. These two parameters are interesting because they allow us to easily compare the five crossover operators only by doing the average of the deviations (find them at the bottom of the tables).

D=15					
C	**C0**	**C1**	**C2**	**C3**	**C4**
Best result	2235	2235	2238	2234	2234
Average result	2246	2235	2360	2235	2237
Deviation respect best result	0,04%	0,04%	4,21%	0,00%	0,00%
Deviation respect best average result	0,48%	0,03%	5,59%	0,00%	0,00%
D=18					
C	**C0**	**C1**	**C2**	**C3**	**C4**
Best result	2426	2426	2626	2450	2445
Average result	2465	2448	2653	2461	2455
Deviation respect best result	0,00%	0,00%	8,24%	0,99%	0,78%
Deviation respect best average result	0,69%	0,00%	8,35%	0,50%	0,27%
D=20					
C	**C0**	**C1**	**C2**	**C3**	**C4**
Best result	2615	2583	2833	2594	2590
Average result	2632	2613	2882	2617	2626
Deviation respect best result	1,24%	0,00%	9,68%	0,43%	0,27%
Deviation respect best average result	0,73%	0,00%	10,31%	0,15%	0,51%
D=22					
C	**C0**	**C1**	**C2**	**C3**	**C4**
Best result	2770	2744	3104	2764	2750
Average result	2811	2780	3145	2790	2773
Deviation respect best result	0,95%	0,00%	13,12%	0,73%	0,22%
Deviation respect best average result	1,38%	0,26%	13,44%	0,63%	0,00%
D=25					
C	**C0**	**C1**	**C2**	**C3**	**C4**
Best result	3008	2936	3397	2952	2962
Average result	3020	2947	3435	2983	2983
Deviation respect best result	2,45%	0,00%	15,70%	0,54%	0,89%
Deviation respect best average result	2,47%	0,00%	16,54%	1,22%	1,20%
Total					
Average deviation respect best result	0,94%	0,01%	10,19%	0,54%	0,43%
Average deviation respect best average result	1,15%	0,06%	10,85%	0,50%	0,41%

Table 5.4: Small dimension 4WA results

So what we can first observe is that all the studied crossover operators work relatively well except for C2 which is a 10% worse than the others according to the average deviations. In particular C4 is the best for WA=1 tests and C1 is the best for WA=4 tests. However the performance is really similar between these two crossovers and also with C0 and C3, so we need to see how they perform with bigger problems and if these small differences grow proportionally or not.

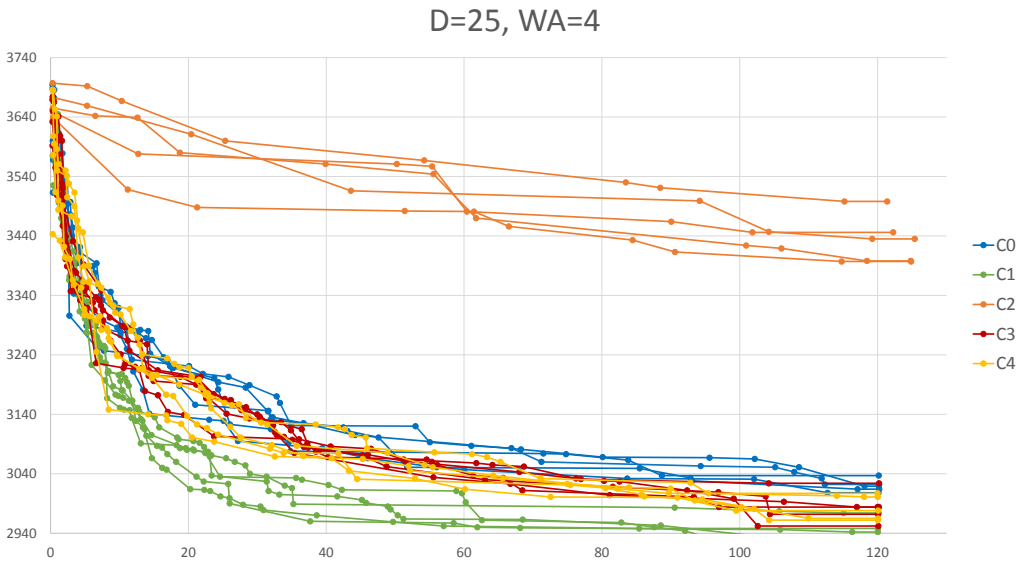

Figure 5.3: Evolution of the different crossover operators with a D=25 and WA=4 problem

To center the further tests we can already make a first selection. We discard C2 for being the worst crossover by far. Also, because C4 is a direct version of C3 and it performs slightly better in all cases we can definitely chose C4 and discard C3. Finally we will also discard C0 even though it gets similar results than C1 and C4 because for both cases with 1 and 4 *WA* either C1 or C4 are in all better. So in conclusion, we select C1 and C4 for the further tests.

5.3.3 Step 3-Big dimensions. Which is the best crossover operator?

Having selected C1 and C4, at this point we keep testing with bigger dimensions, D=30,40 and 50. Because we only have now 2 crossover operators we propose a third one which is a combination of C1 and C4 called C14. The new crossover operator C14 consists in using on each iteration either C1 or C4 randomly. With this new crossover operator we might get more variety on the population without losing quality. We keep all the other parameters, that is, WA=1 and 4, N=5, and a running time of 120 seconds.

In tables 5.5 and 5.6 we can observe the same kind of data than in the previous tables but with some extra information. The average improvement represents the average improvement on the objective function for the five solutions of each

D=30			
C	**C1**	**C4**	**C14**
Best result	2920	2939	2970
Average result	2953	3043	3065
Average Improvement	33,19%	32,33%	31,83%
Deviation respect best result	0,00%	0,65%	1,71%
Deviation respect best average result	0,00%	3,04%	3,79%
Deviation respect average improvement	0,00%	2,59%	4,09%
D=40			
C	**C1**	**C4**	**C14**
Best result	3860	3871	3676
Average result	3911	4010	3864
Average Improvement	31,36%	30,77%	32,89%
Deviation respect best result	5,01%	5,30%	0,00%
Deviation respect best average result	1,21%	3,79%	0,00%
Deviation respect average improvement	4,65%	6,46%	0,00%
D=50			
C	**C1**	**C4**	**C14**
Best result	4784	4941	4750
Average result	4854	5064	4981
Average Improvement	30,93%	27,92%	29,97%
Deviation respect best result	0,72%	4,02%	0,00%
Deviation respect best average result	0,00%	4,34%	2,63%
Deviation respect average improvement	0,00%	9,72%	3,11%
Total			
Average deviation respect best result	1,91%	3,33%	0,57%
Average deviation respect best average result	0,40%	3,72%	2,14%
Total Average Improvement	31,83%	30,34%	31,56%
Average deviation respect best average of improvement	1,55%	6,25%	2,40%

Table 5.5: Big dimension 1WA results

D=30			
C	**C1**	**C4**	**C14**
Best result	3359	3324	3329
Average result	3391	3355	3353
Average Improvement	19,02%	20,32%	19,97%
Deviation respect best result	1,05%	0,00%	0,15%
Deviation respect best average result	1,15%	0,07%	0,00%
Deviation respect average improvement	6,37%	0,00%	1,72%
D=40			
C	**C1**	**C4**	**C14**
Best result	4003	4093	4018
Average result	4069	4171	4084
Average Improvement	20,09%	19,64%	21,70%
Deviation respect best result	0,00%	0,25%	0,37%
Deviation respect best average result	0,00%	2,53%	0,37%
Deviation respect average improvement	7,42%	9,49%	0,00%
D=50			
C	**C1**	**C4**	**C14**
Best result	4793	5006	4860
Average result	4875	5056	4966
Average Improvement	23,23%	18,92%	20,57%
Deviation respect best result	0,00%	4,44%	1,40%
Deviation respect best average result	0,00%	3,70%	1,87%
Deviation respect average improvement	0,00%	18,57%	11,44%
Total			
Average deviation respect best result	0,35%	2,23%	0,64%
Average deviation respect best average result	0,38%	2,10%	0,75%
Total Average Improvement	20,78%	19,62%	20,75%
Average deviation respect best average of improvement	4,60%	9,35%	4,39%

Table 5.6: Big dimension 4WA results

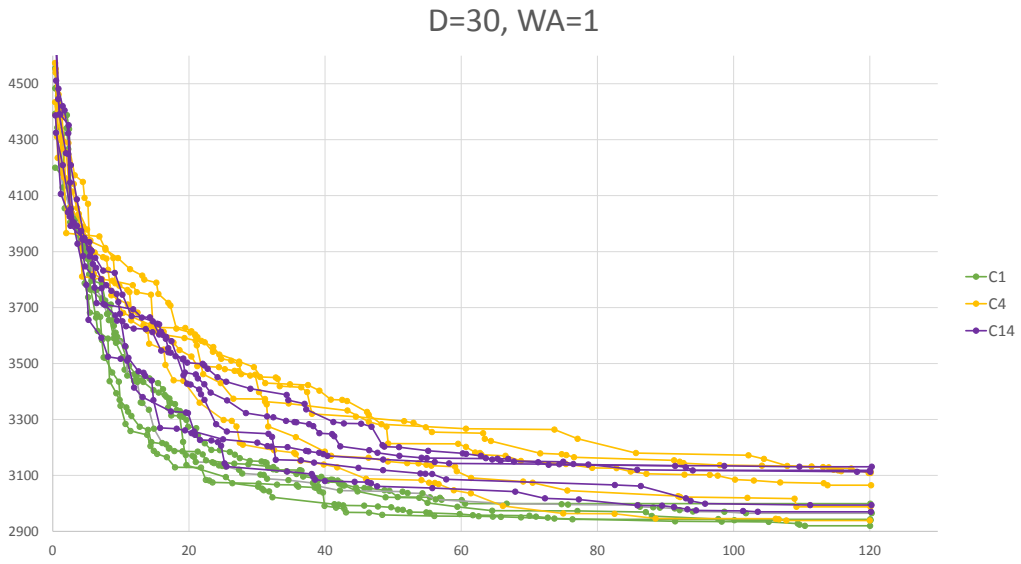

Figure 5.4: Evolution of the different crossover operators with a D=30 and WA=1 problem

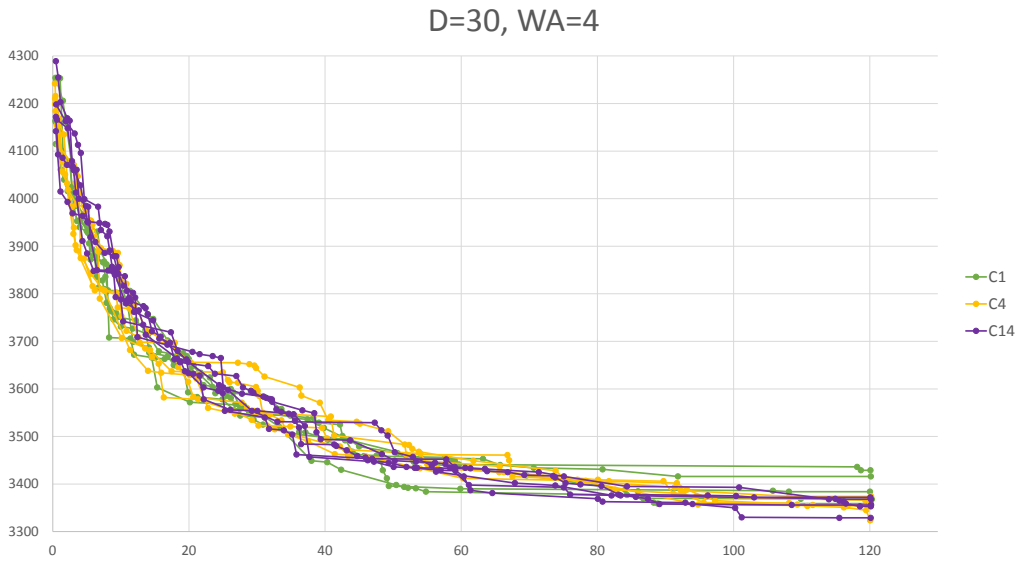

Figure 5.5: Evolution of the different crossover operators with a D=30 and WA=4 problem

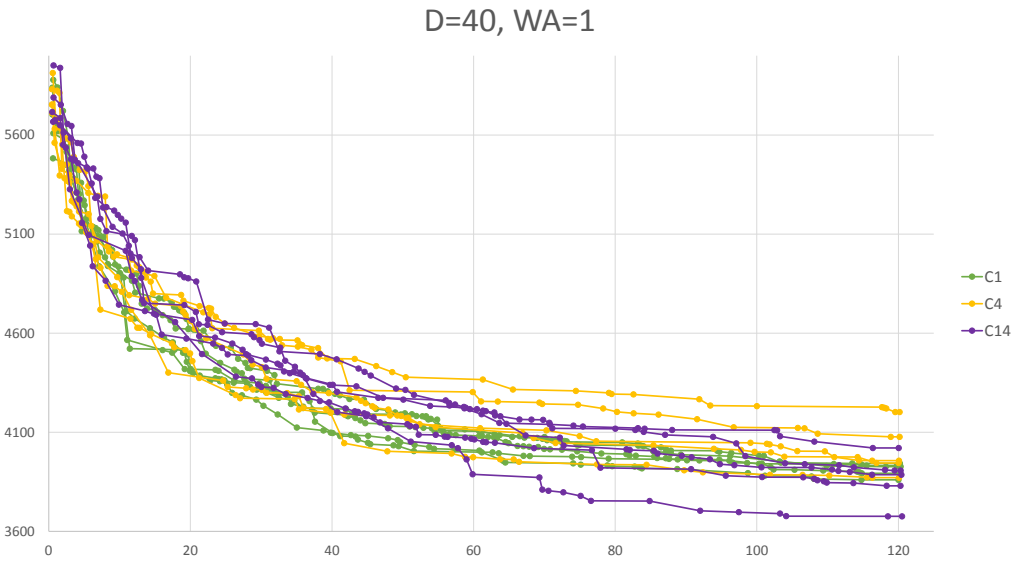

Figure 5.6: Evolution of the different crossover operators with a D=40 and WA=1 problem

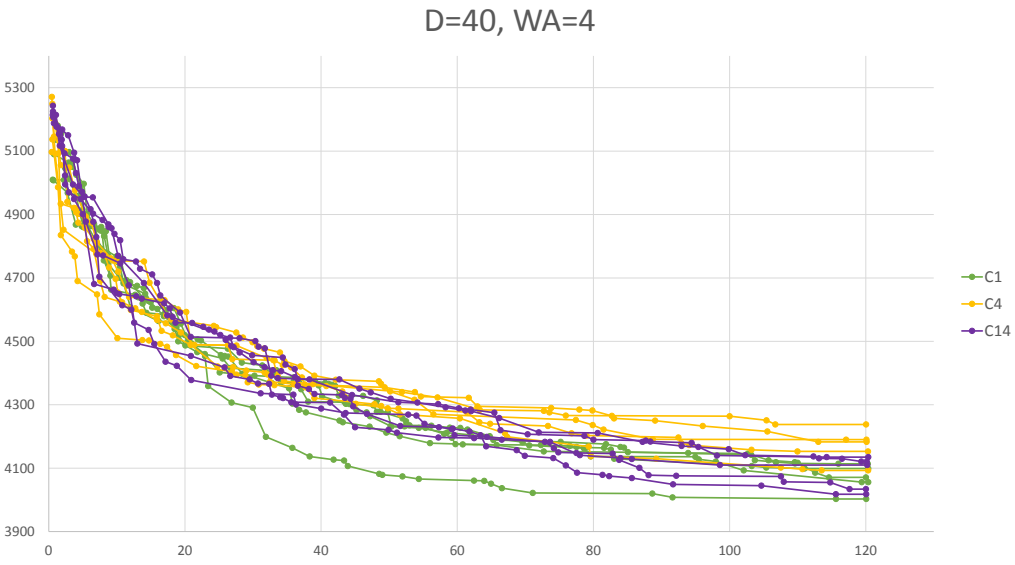

Figure 5.7: Evolution of the different crossover operators with a D=40 and WA=4 problem

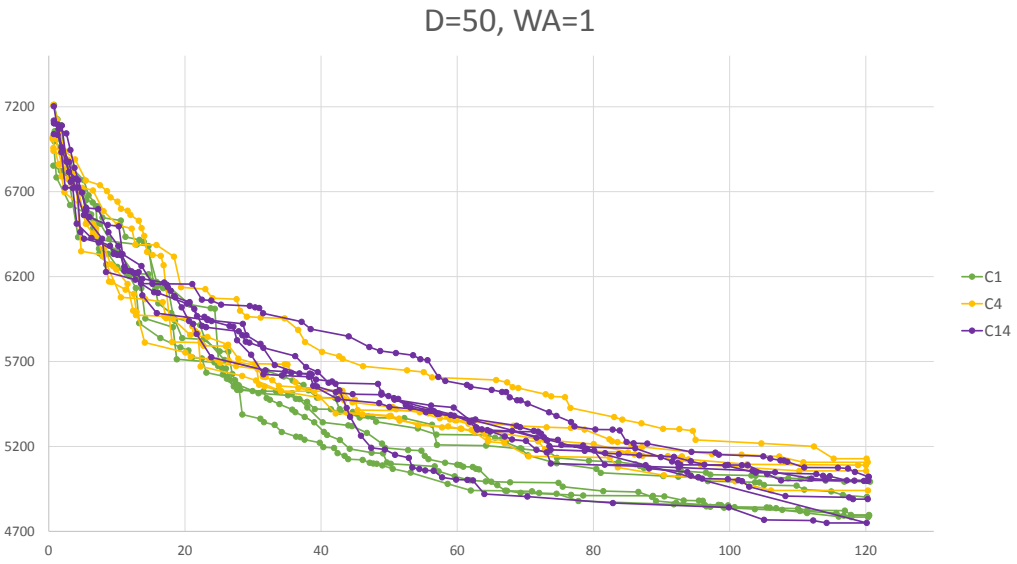

Figure 5.8: Evolution of the different crossover operators with a D=50 and WA=1 problem

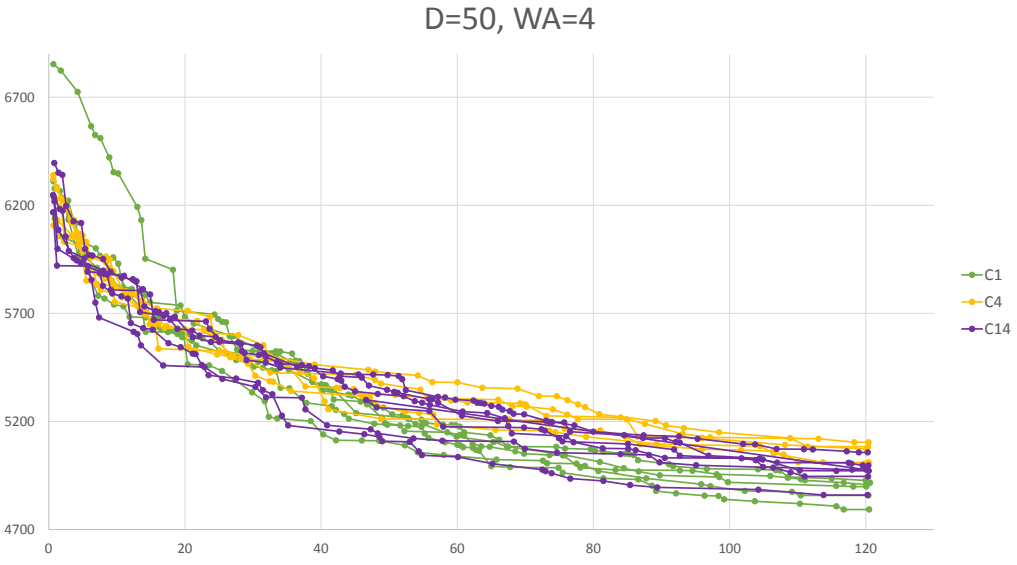

Figure 5.9: Evolution of the different crossover operators with a D=50 and WA=4 problem

case, that is the initial value minus the final value relative to the initial value. The deviation respect average improvement is the relative distance between the best average improvement obtained with each crossover and the best result of average improvement among all 3 crossovers. Similar to the previously explained parameters, the total average improvement is the average for each crossover operator of the average improvement for the dimensions 30, 40 and 50, and the average deviation respect best average of improvement is the average also for each crossover operator of the deviation respect average improvement.

Also in Figures 5.4, 5.5, 5.6, 5.7, 5.8 and 5.9 we can observe again the evolution of the different tests with the completion time in y and the running time in x (set up to 120s). Remember that each point represents a new best value.

The information shown in the graphics does not allow us to draw clear conclusions. For example in the D=30 and WA=1 test (Figure 5.4) we can observe that C1 works better than C4 and C14, but then this difference is not seen in the D=40 (Figure 5.6) and D=50 (Figure 5.8) cases. All three crossover operators seem to work in a similar way, at least from a first visual approach.

If we analyze the data we find that the difference in performance between the three crossover operators is indeed minimum. The average deviation respect best result for C1 and C14 in either WA=1 (Table 5.5) and WA=4 (Table 5.6) tests is always less than 2% and the total average improvement is between 30% and 32% for WA=1 and between 19% and 21% for WA=4 in all cases. Knowing how these crossover operators work, it seems reasonable that a big part of these small differences are due to the randomness involved.

Remember that these so called big dimensions tests were proposed to determine whether the minimum differences shown between the crossover operators C1 and C4 in the small dimension tests would became more relevant as the problem dimension was increased. However the data showed that even though these differences are slightly higher for big dimensions tests, they are still minimal.

We propose then to select between C1, C4 and C14 by simply looking at which one has more total better results (again in Tables 5.5 and 5.6). And this happens to be C1 which has, between the WA=1 and WA=4 tests, 6 better results out of 8 possible ones.

5.4 Parameters calibration

The goal of the previous tests was to select one crossover operator for the algorithm. For that we fixed all the other parameters. In particular we have been always using a population size of 100 creatures and in each iteration we

generated 40 new creatures from the crossover and 10 from the mutation process. In Figure 5.10 we can observe a visual representation of this intermediate algorithm used until now:

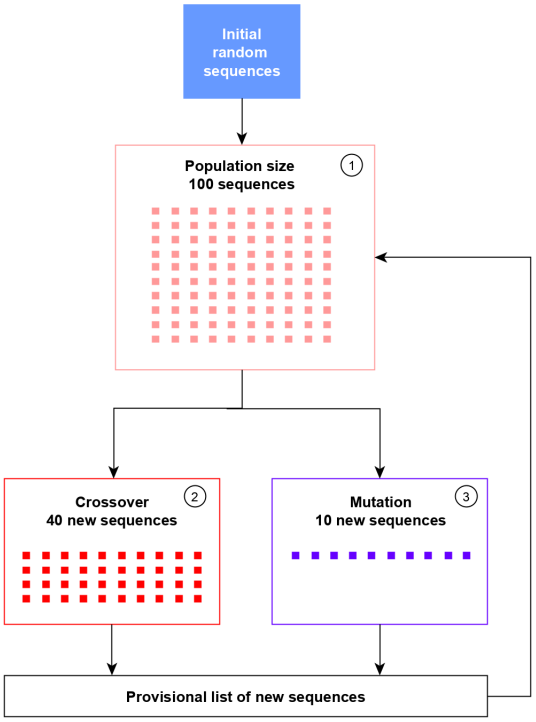

Figure 5.10: Visual representation of the intermediate genetic algorithm used in the previous tests

The points (1), (2) and (3) in Figure 5.10 represent the different parameters that have to be adjusted for the final version. Note that in (3) there is a double parameter:

1. Population size or **PS**
2. Number of Crossover creatures or **NCc**. This is the number of generated creatures from the crossover in each iteration.
3. Number of Mutant creatures or **NMc**. This is the number of generated creatures from mutation in each iteration.
4. Mutation method or **Mm** which is the operating process of the mutation itself.

For the population size we consider 3 different values: 50, 100 (the one used until now) and 200. For the NCc we also consider three different values: 25%, 50%

and 75%. This values are relative to the PS because we want to isolate the NCc effect with respect to the PS. Therefore a NCc value of 25% with a PS of 100 creatures means that on every iteration 25 new creatures will be created from the crossover operator. Similarly, for the NMc we consider values of 5%, 10% and 25% but also a special variable value. Note that until now all the shown values are constant. Some studies instead have considered variable mutation parameters, also called augmented parameters with an evolutionary scheme, because the idea is to let them change according to the evolutionary process. For example R. Cazacu [3] concluded that the evolution scheme of the mutation parameter is beneficial from both the reliability and the accuracy point of view. Then we will consider also a variable option that will start at 5% and will grow in every iteration +1% if the the best 10 sequences have not changed since the last iteration or will decrease -1% if at least one of the best 10 sequences have changed since the last iteration. These variations will be limited by a lower bound and an upper bound set up at 5% and 80%. With this variable option we help the algorithm to explore a local area when the solution is improving but at the same time to explore different areas of the space of solutions when it gets stuck in a local optimum.

Finally we have considered 4 different Mutation methods or Mm. We will use the same names that Soni and Kumar used in their study of various mutation operators in genetic algorithms [17]:

- **Swap Mutation**: Select two nodes at random and swap their positions.
- **Inverse Mutation**: Select two nodes at random and invert the substring between them.
- **Insert Mutation**: Select two nodes at random and move the second one to follow the first one, shifting the rest along to accommodate.
- **Scramble Mutation**: Select a subset of nodes at random and then randomly rearrange them also at random in those positions. Subset does not have to be contiguous.

For the travelling salesman problem (TSP) the Inverse mutation (also Known as Reverse Sequence Mutation or RSM) seems to be the mutation operator with better performance among the ones presented above [13]. However like the crossover operator, this depends specifically on the problem characteristics. This is why we consider at first multiple options.

To sum up, all the different considered values for the 4 described parameters are shown in Table 5.7:

With these parameter values there are $3x3x4x4 = 140$ combinations. Because there is a lot of randomness involved we will solve again each combination of parameters 5 times, which makes then a total of 720 cases. We set up the calculation time to 100s per case so in total the calibration tests will take 72.000 seconds or what it is the same 20 hours.

PS	NCc	NMc	Mm
50	25%	5%	swap
100	50%	10%	inverse
200	75%	25%	insert
		Variable	scramble

Table 5.7: Values considered for the different parameters

Note that we reduce the calculation time per case from 120 to 100 seconds to reduce the total running time of the test having seen previously that during these last 20 seconds the solutions do not improve considerably.

We analyze the results with *Minitab* software using the factorial regression design and we find that that Mm and the interaction between PS-NMc and PS-Mm are the most influential parameters. We can see that on the Pareto chart of standardized effects shown in Figure 5.11.

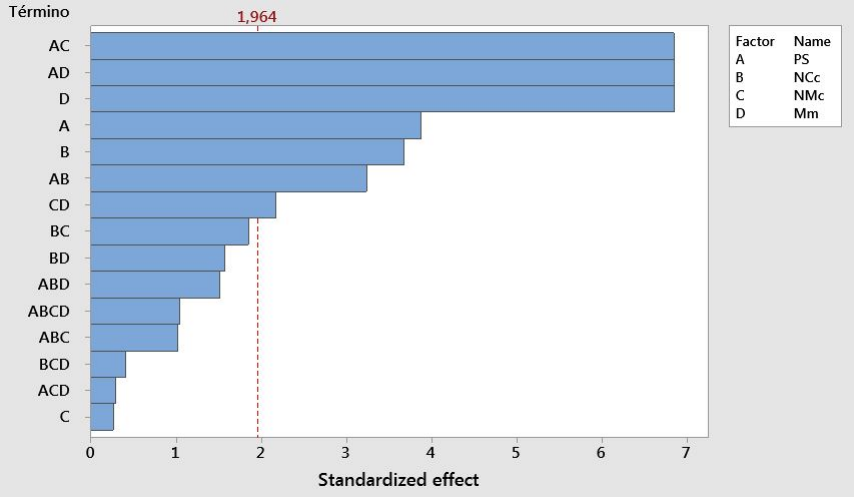

Figure 5.11: Pareto chart of standardized effects with a 95% confidence interval

The isolated effects of the four parameters also called main effects in the statistics field of study are shown in Figure 5.12 in which the makespan average is shown for every parameter value. Note that the NMc levels, *low*, *medium* and *high* refer to the 5%, 10%, and 25% values. We can check that indeed the Mm is the factor with highest impact on the makespan, with best results found with the insert and swap mutation methods. The makespan also improves as the NCc decreases and PS obtains best results with 100 creatures.

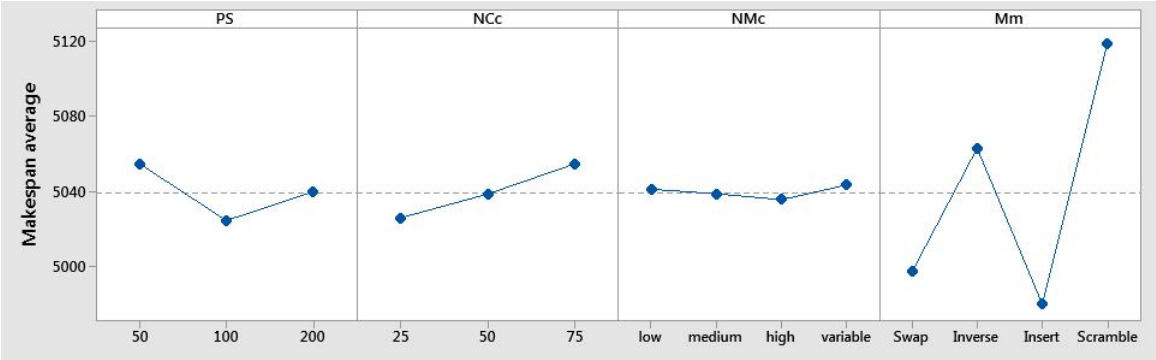

Figure 5.12: Chart of main effects

The next step is to look at the interaction between the parameters, specially between PS and Mm, and PS and NMc as they were considered on the Pareto Chart as the most influentials. These interactions are shown in Figure 5.13. There, we can observe on the PS-NMc interaction that the best results are obtained with PS=50 or 100 and high NMc or PS=200 and low NMc. This interaction effect makes actually a lot of sense as it basically tells us that the best results are obtained when the calculation time per iteration does not grow too much. The higher the PS and the NMc are, the higher also the calculation time per iteration is. So in order to keep the calculation time per iteration balanced when one parameter increases the other has to decrease. And this is what we can observe on the PS-NMc interaction effect graph.

On the other hand the PS-Mm interaction shows us that the higher the PS the less important is the Mm. So with PS=50 the difference of performance between the best and the worst Mm (Insert and Scramble) is much higher than with PS=200. Also with PS=200 the results get considerably worse than with PS=50 or 100.

To fix the parameters for the final algorithm we begin by choosing a population size (**PS**) of 100 creatures and a number of generated creatures per iteration from the crossover of 25 (**NCc**) as they are clearly the best values for the main effects respectively. Then we chose the insert mutation method (**Mm**) as it is the best interaction for a PS=100. And finally we have the NMc that in the Pareto Chart (Figure 5.11) is shown as the less important of the effects. For this reason we simply chose the variable value of **NMc** or generated creatures per iteration from mutation operator. Remember that the variable NMc value will start at 5% and will grow or decrease in every iteration $\pm 1\%$ if the best 10 sequences have or have not changed since the last iteration. The chosen values for the four studied parameters are summarized in Table 5.8.

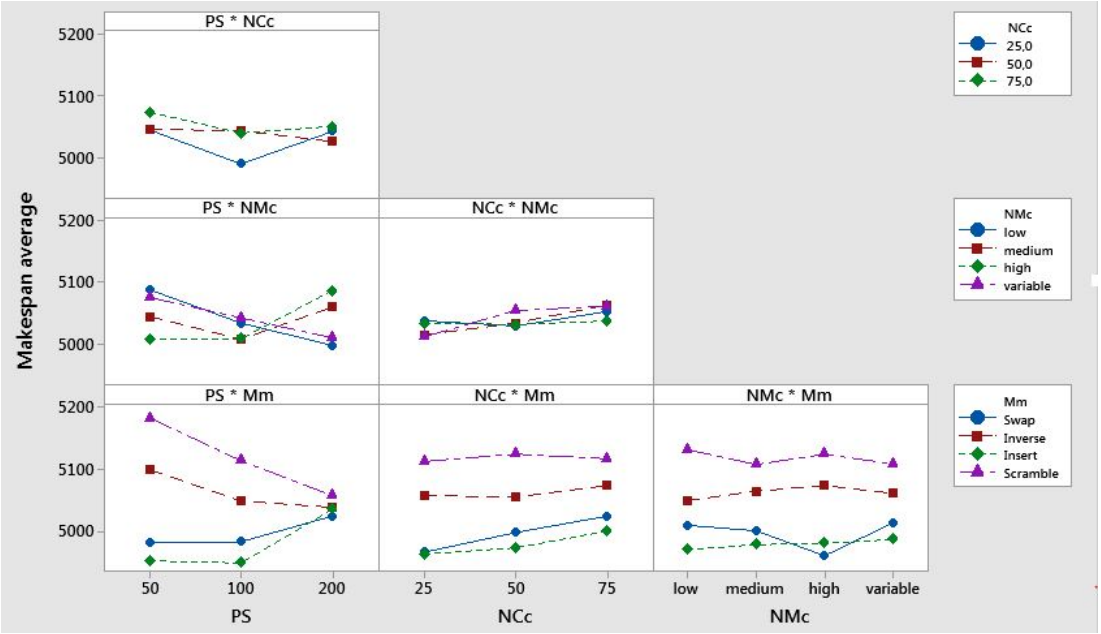

Figure 5.13: Chart of interaction effects

PS	**NCc**	**NMc**	**Mm**
100	25%	variable	Insert

Table 5.8: Chosen values for the different studied parameters

5.5 Final results

At this point we just have to compare the final genetic algorithm with the initial one (Chapter 4) to see if there is any improvement. To do so we will solve 100 randomly generated different cases with different number of pieces N, number of machine stations M and waiting area stations WA. Each case will be solved 4 times to reduce the effect of randomness. Also we will add to the comparison the intermediate algorithm before the parameter calibration of section 5.4 to check if not only the crossover operator selection but also the calibration of PS, NCc, NMc and Mm did any improvement. Ideally we would expect to see an improvement on both stages.

Considering that we want to solve 100 cases with 3 different algorithms and 4 times each and that for each case the running time is set to 120s, the total running time of the test will be 2400 minutes or 40 hours.

In Table 5.9 we can find for each of the 100 different cases and each algorithm (Initial, Intermediate and Final), the average makespan for the four obtained results. I_1 represents the improvement between the initial and the intermediate algorithm, that is: $\frac{Initial-Intermediate}{Initial}(\%)$. Similarly I_2 measures the improvement between the intermediate and the final algorithm and I_{total} the improvement between the initial and the final algorithm.

Case	Initial	Intermediate	Final	I_1	I_2	I_{total}
1	6015,5	5827,5	5928	3,13%	-1,72%	1,45%
2	2765,5	2682	2620	3,02%	2,31%	5,26%
3	2707,25	2801,25	2708	-3,47%	3,33%	-0,03%
4	3412	3372,25	3325	1,17%	1,40%	2,55%
5	4666,75	5242	5119,25	-12,33%	2,34%	-9,70%
6	3039,75	3203	3193	-5,37%	0,31%	-5,04%
7	4265,5	4200	4159,75	1,54%	0,96%	2,48%
8	3244,25	3326,5	3285	-2,54%	1,25%	-1,26%
9	3157	3024,5	2960	4,20%	2,13%	6,24%
10	2444,75	2470,75	2411,5	-1,06%	2,40%	1,36%
11	5909,5	5904,5	5899,25	0,08%	0,09%	0,17%
12	5170,25	5046	5013,5	2,40%	0,64%	3,03%
13	2702,75	2613,25	2601,25	3,31%	0,46%	3,76%
14	4381,5	4076	4081,75	6,97%	-0,14%	6,84%
15	3436	3286,25	3258,25	4,36%	0,85%	5,17%
16	5060,25	4851,25	4803,5	4,13%	0,98%	5,07%
17	3836,75	3705,25	3699,75	3,43%	0,15%	3,57%
18	4314,75	4023,75	3893,5	6,74%	3,24%	9,76%
19	2236,25	2139,25	2153	4,34%	-0,64%	3,72%
20	4527,25	4328,25	4263	4,40%	1,51%	5,84%
21	3162,25	2985,5	2932,5	5,59%	1,78%	7,27%
22	4065,75	4643,25	4390,75	-14,20%	5,44%	-7,99%
23	3649,25	3855,75	3734,75	-5,66%	3,14%	-2,43%
24	4343,5	4248,5	4209,75	2,19%	0,91%	3,08%
25	2683,5	2849,25	2645	-6,18%	7,17%	1,43%
26	4946,75	4896,5	4853,5	1,02%	0,88%	1,89%
27	2071,5	1997,5	1965,25	3,57%	1,61%	5,13%
28	2597	2636	2564,5	-1,50%	2,71%	1,25%
29	4176,75	3982,75	4018	4,64%	-0,89%	3,80%
30	2542,75	2493	2465,5	1,96%	1,10%	3,04%
31	3631,75	3419	3384	5,86%	1,02%	6,82%
32	2354,75	2300,75	2231,75	2,29%	3,00%	5,22%
33	2478,25	2415,5	2387	2,53%	1,18%	3,68%

Case	Initial	Intermediate	Final	I_1	I_2	I_{total}
34	3911,5	3831,5	3763	2,05%	1,79%	3,80%
35	2765,75	2710	2635,5	2,02%	2,75%	4,71%
36	4345,25	4317	4290,75	0,65%	0,61%	1,25%
37	3512	3341	3377	4,87%	-1,08%	3,84%
38	3837,5	4041	4040,5	-5,30%	0,01%	-5,29%
39	4008,5	3834,75	3853,75	4,33%	-0,50%	3,86%
40	4286,75	4314,5	4257,5	-0,65%	1,32%	0,68%
41	2925,75	2852,5	2800,75	2,50%	1,81%	4,27%
42	3006,75	2952	2912,75	1,82%	1,33%	3,13%
43	3436,75	3337,5	3299,25	2,89%	1,15%	4,00%
44	5841,25	5400,5	5498,25	7,55%	-1,81%	5,87%
45	4240,25	4161,5	4101,5	1,86%	1,44%	3,27%
46	5245,75	5217	5177,5	0,55%	0,76%	1,30%
47	2547,5	2473,5	2481,75	2,90%	-0,33%	2,58%
48	5197,5	5036,75	4966,5	3,09%	1,39%	4,44%
49	2366	2334	2313,25	1,35%	0,89%	2,23%
50	2006,25	1950,5	1980,75	2,78%	-1,55%	1,27%
51	3583,25	3478,75	3421,5	2,92%	1,65%	4,51%
52	3863,75	3819	3784,5	1,16%	0,90%	2,05%
53	2508	2430,25	2433,25	3,10%	-0,12%	2,98%
54	2574,5	2527,25	2502,75	1,84%	0,97%	2,79%
55	4546	4664	4633,75	-2,60%	0,65%	-1,93%
56	2228,75	2101,25	2085,75	5,72%	0,74%	6,42%
57	2529,75	2440,75	2434,25	3,52%	0,27%	3,78%
58	3558,25	3488,5	3442,75	1,96%	1,31%	3,25%
59	3294,5	3260,5	3242,5	1,03%	0,55%	1,58%
60	4383,5	4377,25	4276,25	0,14%	2,31%	2,45%
61	2343,75	2154,25	2117,5	8,09%	1,71%	9,65%
62	3111	3029	2968,5	2,64%	2,00%	4,58%
63	4022,5	3871,5	3907	3,75%	-0,92%	2,87%
64	3530,25	3506,5	3451,25	0,67%	1,58%	2,24%
65	2454,5	2408,75	2393,5	1,86%	0,63%	2,49%
66	5563	5338	5283,75	4,04%	1,02%	5,02%
67	5351,75	5193,25	5196,5	2,96%	-0,06%	2,90%
68	4347	4563	4495,25	-4,97%	1,48%	-3,41%
69	5449	5401,75	5394,5	0,87%	0,13%	1,00%
70	3910,25	3789,25	3784,25	3,09%	0,13%	3,22%
71	2484	2457,75	2469	1,06%	-0,46%	0,60%
72	3708,25	3655,25	3557,75	1,43%	2,67%	4,06%
73	5014,25	4936,25	4925,5	1,56%	0,22%	1,77%
74	5283	5268	5094,5	0,28%	3,29%	3,57%
75	1999,25	1924,75	1932	3,73%	-0,38%	3,36%
76	3029,75	2950	2919,5	2,63%	1,03%	3,64%

Case	Initial	Intermediate	Final	I_1	I_2	I_{total}
77	4989,5	4851,5	4934	2,77%	-1,70%	1,11%
78	4188,75	4462,75	4384,75	-6,54%	1,75%	-4,68%
79	5841,25	5637	5704	3,50%	-1,19%	2,35%
80	3781,25	3774,25	3741,75	0,19%	0,86%	1,04%
81	5081	5060	5108,5	0,41%	-0,96%	-0,54%
82	2809	2750,75	2725,5	2,07%	0,92%	2,97%
83	3890,5	3893,75	3936,25	-0,08%	-1,09%	-1,18%
84	5189,75	5147,5	5071,25	0,81%	1,48%	2,28%
85	3698,25	3636,75	3581,5	1,66%	1,52%	3,16%
86	4558,75	4543,5	4525	0,33%	0,41%	0,74%
87	4135	3925,25	3897	5,07%	0,72%	5,76%
88	3880,75	3654,75	3560,25	5,82%	2,59%	8,26%
89	2612,25	2519	2510,5	3,57%	0,34%	3,90%
90	5666,25	5359,75	5361	5,41%	-0,02%	5,39%
91	4324,25	3968,5	3837,25	8,23%	3,31%	11,26%
92	3812	3626	3597,5	4,88%	0,79%	5,63%
93	5253,5	5026	5004,25	4,33%	0,43%	4,74%
94	6260	6625,75	6560,25	-5,84%	0,99%	-4,80%
95	2953,75	3015,25	2953	-2,08%	2,06%	0,03%
96	4542,25	4597	4553,75	-1,21%	0,94%	-0,25%
97	3874	3797,25	3764,25	1,98%	0,87%	2,83%
98	2295,25	2145,25	2110,5	6,54%	1,62%	8,05%
99	3044	3042	2942	0,07%	3,29%	3,35%
100	4704,25	4822,25	4754	-2,51%	1,42%	-1,06%
TOTAL				**1,58%**	**1,06%**	**2,64%**

Table 5.9: Comparison of the different algorithm versions performance

The final genetic algorithm is 2,64% better than the initial one which is mainly achieved by the crossover operator selection (represented by I_1) with an improvement of 1,58%. The final calibration (I_2) shows also an improvement but a bit lower than I_1, of 1,06%. This is probably because the final values of the calibration process resulted to be accidentally quite similar than the ones already used. On the initial and intermediate algorithms the population size was fixed to 100 creatures and the mutation method used was the scramble one. And indeed, after the calibration process we selected for the final version a population size of 100 creatures and the insert mutation method that was very similar with the scramble method and actually very close on performance.

A total improvement of 2,64% is therefore very positive for two reasons. On the one hand because this improvement is the average of 100 different cases that at the same time were solved 4 times each, so that the effect of randomness was strongly reduced. On the other hand because the calibration showed that

the initial parameter values were, by change, quite close to the optimum ones which means that if those initial values would have been different the improvement would have been much higher.

We also notice that despite the general improvement there are 13 cases (3, 8, 11, 22, 23, 38, 55, 68, 78, 83, 94, 96 and 100) that instead of improving, they get a worse completion time with the final algorithm. We can not detect any pattern on these cases, such as that they all have a high number of pieces (N) or machine stations (M) or waiting area stations (H) so the reason of these deteriorations is unknown.

Chapter 6

Conclusion

The main goal of this thesis was to design and code a genetic algorithm that improve the performance of the initial algorithm.

To do so we first focused on the crossover operator as we considered it the most challenging part on a sequencing problem. We studied and compared 5 different crossover operators and we found that 2 of them performed better than the others: the Cycle Crossover (CX) and the Union Crossover 2 version 2 (UX2 v2) a version of the UX2 that we developed thinking that it could improve its performance for this problem. Finally we selected the CX.

Then we calibrated the other parameters that we considered important: The population size, the weight of mutation and crossover processes on the algorithm and the mutations operator itself. We discovered that the optimum values for those parameters were not very far from the values already used on the initial and intermediate version. That is a population size of 100 creatures, a number of generated creatures from the crossover operator per itarion of 25%, a variable value for the number of generated creatures from the mutation operator per iteration that depended on the improvement or not of the best 10 sequences on each iteration and the Insert mutation operator.

With all this we finally compared the performance of the final algorithm with the initial one and also with the intermediate one, the one just after the crossover operator selection, to see if both the crossover operator and the final calibration brought some improvement to the algorithm. The results showed an improvement as expected on both stages so that the average total improvement was 2,64%.

Bibliography

[1] Zakir H Ahmed. *Genetic Algorithm for the Traveling Salesman Problem Using Sequential Constructive Crossover Operator*. International Journal of Biometrics Bioinformatics (IJBB) Volume (3): Issue (6), 2010.

[2] A. Allahverdi. *A survey of scheduling problems with no-wait in process*. European Journal of Operational Research Volume 255, Issue 3, 2016.

[3] R. Cazacu. *Comparative Study between the Improved Implementation of 3 Classic Mutation Operators for Genetic Algorithms*. Procedia Engineering 181 (2017) Pages 634 – 640, 2017.

[4] David E.Goldberg. *Genetic Algorithms in search, Optimization and Machine Learning*. Pearson, 2006.

[5] B. R. Fox and M. B. McMahon. *Genetic operators for sequencing problems*. Kaufmann pp. 284-300., 1991.

[6] G.Vairaktarakis H.Emmons. *Flow Shop Scheduling. Theoretical results, algorithms and applications*. International Series in Operations Research Managment Science. Volume 182, 2013.

[7] D. J. d. Smith I. M. Oliver and R. C. J. Holland. *A study of permutation crossover operators on the traveling salesman problem*. L. Erlbaum Associates Inc., 1987.

[8] J.Pempera J.Grabowski. *Sequencing of jobs in some production system*. European Journal of Operational Research. Volume 125, Issue 3, pp 535-550, 2000.

[9] S. M. Johnson. *Optimal two- and three-stage production schedules with setup times included*. RAND Corporation, 1953.

[10] D.Fuquay L.Darrell Whitley, T.Starkweather. *Scheduling Problems and Traveling Salesmen: The Genetic Edge Recombination Operator*. Morgan Kaufmann Publishers Inc., 1989.

[11] C.Sriskandarajah N.G.Hall. *A survey of machine scheduling problems with blocking and no-wait in process.* Operations Research. Volume 44, no. 3, pp. 510-525, 1996.

[12] A.B.Chandramouli N.Tyagi, R.P.Tripathi. *A New Heuristic Algorithm for Four Machines Stochastic Flow Shop Scheduling Model.* 2017 IEEE 3rd International Conference on Advances in Computing,Communication Automation, 2018.

[13] J. Abouchabaka O. Abdoun and C. Tajani. *Analyzing the Performance of Mutation Operators to Solve the Travelling Salesman Problem.* Computing Research Repository, 2012.

[14] P. W. Poo and J. N. Carter. *Genetic algorithm crossover operators for ordering applications.* Computers Ops Res. Vol. 22, No. 1, pp. 135-147, 1995.

[15] X.Cui Q.Zhou. *Research on Multiobjective Flow Shop Scheduling with Stochastic Processing Times and Machine Breakdows.* 2008 IEEE International Conference on Service Operations and Logistics, and Informatics, 2008.

[16] S. S. Reddi and C. V. Ramamoorthy. *On the Flos-shop Sequencing Problem with No Wait in Process.* Pergamon Press 1972. Vol. 23,pp. 323 to 331, 1972.

[17] N. Soni and Dr.Kumar. *Study of Various Mutation Operators in Genetic Algorithms.* International Journal of Computer Science and Information Technologies, Vol. 5 (3) , 2014, 4519-4521, 2014.

[18] R.Ruiz S. Sui Y.Wang, X.Li. *An iterated Greedy Heuristic for Mixed No-Wait Flowshop Problems.* IEEE Transactions on cybernetics. Volume 48, 2018.

[19] C.Shi M.Cui X.Cao Y.Ge Y.Xu, X.Li. *A Methauristic for Mo-wait Flowshops with Variable Processing Times.* Proceedings of the 2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design, 2018.